Distributed Moisture Sensor Network

Matt Burrough
CS 425 – Summer 2012

Introduction

One of my favorite parts of summer is the fresh produce and herbs that bloom in the abundant sun. Unfortunately, the weather during the summer months can be severe – ranging from weeks of drought to days of heavy downpours. As an inexperienced gardener with a full-time job who is also working on a graduate degree, it can become easy to overlook plants for a few days only to find them withered. To avoid this, I propose a collection of sensors to report plants' moisture levels that can be centrally monitored.

This problem is actually an interesting one from a distributed systems perspective. The sensor nodes need to be able to wirelessly report their status on a regular schedule while handling time synchronization, node failures, and a potentially partitioned network. Below, I'll describe the system I built, the relevant design decisions, test results, and potential improvements.

Material Selection and Assembly

There are a variety of sensor node platforms that could be used for this type of project. At the heart of the system, I knew I wanted to use ATMega microcontrollers because of their low cost and their use in the Arduino open source project. The Arduino project is well documented, allows for development of node software in a C-based language, has numerous libraries, and offers a variety of first- and third-party boards that have been thoroughly tested.

Initially, I was considering using a board like the Arduino Nano[1], and paring it with an XBee 802.15.4 RF module[2]. These two boards could be easily connected using a "shield" such as the one available from Robotshop.com[3]. However, together these parts cost approximately $80 for one node, not including tax or shipping. I need at least three nodes (one for the base station and two for plants in different area of my yard), so this proved too costly.

Upon further research, I discovered a lower cost alternative. The JeeLabs[4] JeeNode[5] project is an ATMega/Arduino-based board with an RFM12 radio from HopeRF.[6] These units cost only $22.50 for a kit with all required parts. Additionally, JeeLabs also created JeeLink[7], which is a surface mount ATMega microcontroller, FTDI USB Virtual Serial port chip, and a RFM12 radio in a thumb-drive form factor. The unit comes preassembled for $36.50. These prices were acceptable, so I ordered two JeeNode kits and one JeeLink. All JeeLabs products come with a choice of 434 MHz, 868 MHz (EU only), or 915 MHz (US only) radios. I opted for the 915 MHz versions.

In addition to the microcontroller and radio, the project would not be complete without a means of measuring the moisture in soil. After reviewing various electronics projects site, I found the Grove Moisture Sensor[8], which met my needs precisely. At only $5 each, they were a great choice.

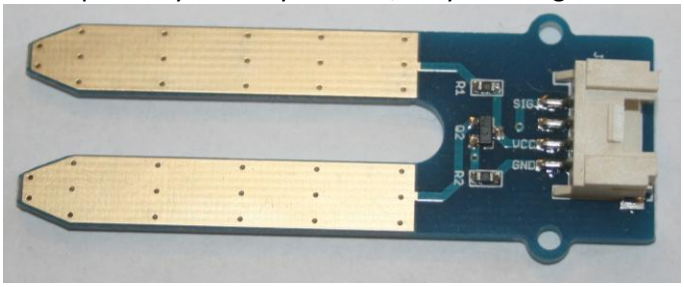

Figure 1: Groove Moisture Sensor

Finally, a few other miscellaneous parts were required:

- Two 3xAAA battery holders[9]
- Six AAA batteries
- Groove wires with connector[10]
- Medium project boxes to hold node boards[11]
- Small project boxes to hold Groove sensors[12]
- Heat shrink tube
- Silicone sealant
- Solder
- FTDI USB to TTL cable to program nodes[13]

After receiving and assembling the JeeNode kits, the next step was to strip the ends of two sets of the grove wires and tin them with solder so they could be inserted into the node's female IO ports. Next, battery holders were soldered to the JeeNodes. Three AAA batteries were chosen as they provide 4.5v of power (an acceptable amount for the JeeNode) in a small and common form factor. Finally, the JeeNodes were placed in project boxes, and the moisture sensors placed in their own boxes, with heat shrink tubing used to protect the wires going between the sensor and the JeeNode. The holes on both ends were plugged with silicon to provide a watertight enclosure for both components.

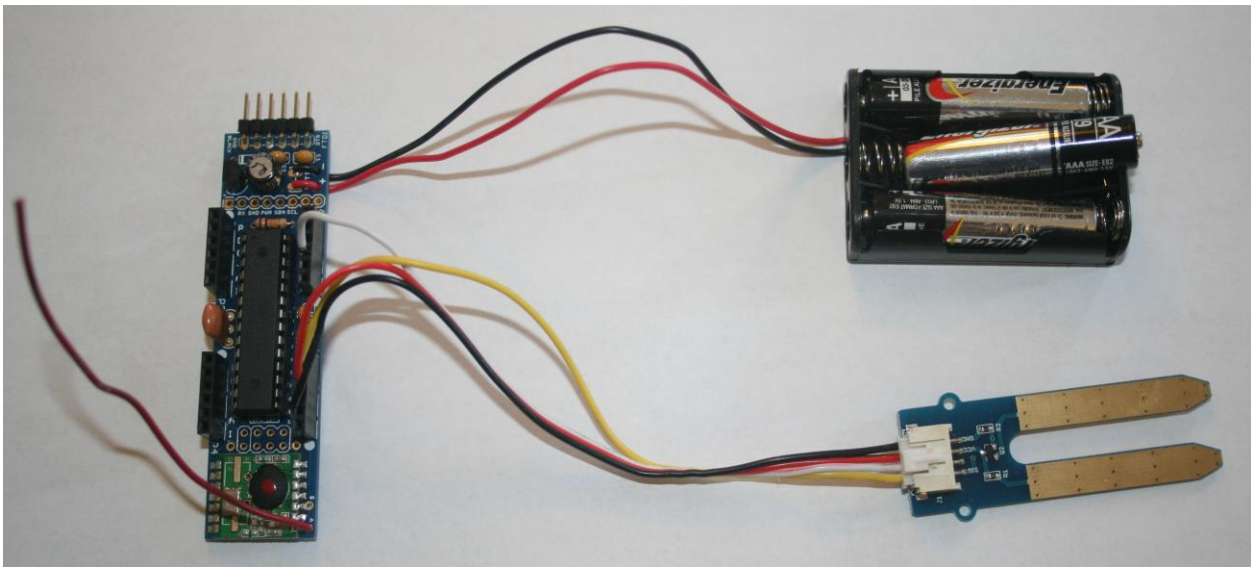

Figure 2: JeeNode with Battery and Groove Moisture Sensor

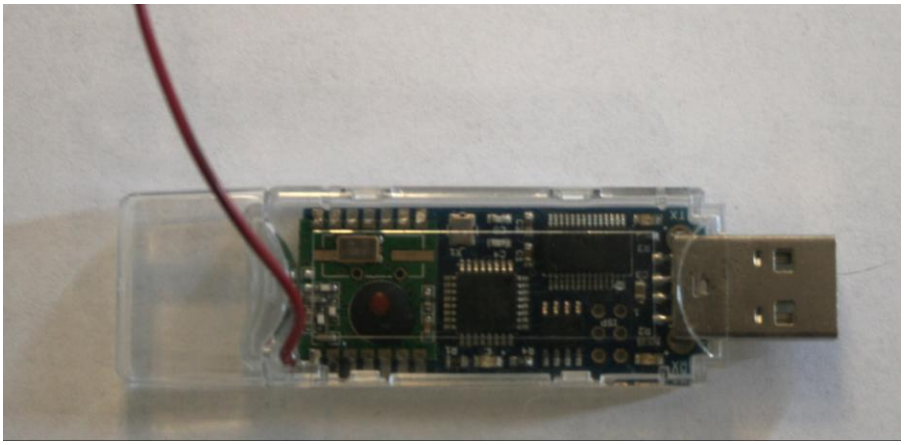

Figure 1: JeeLink

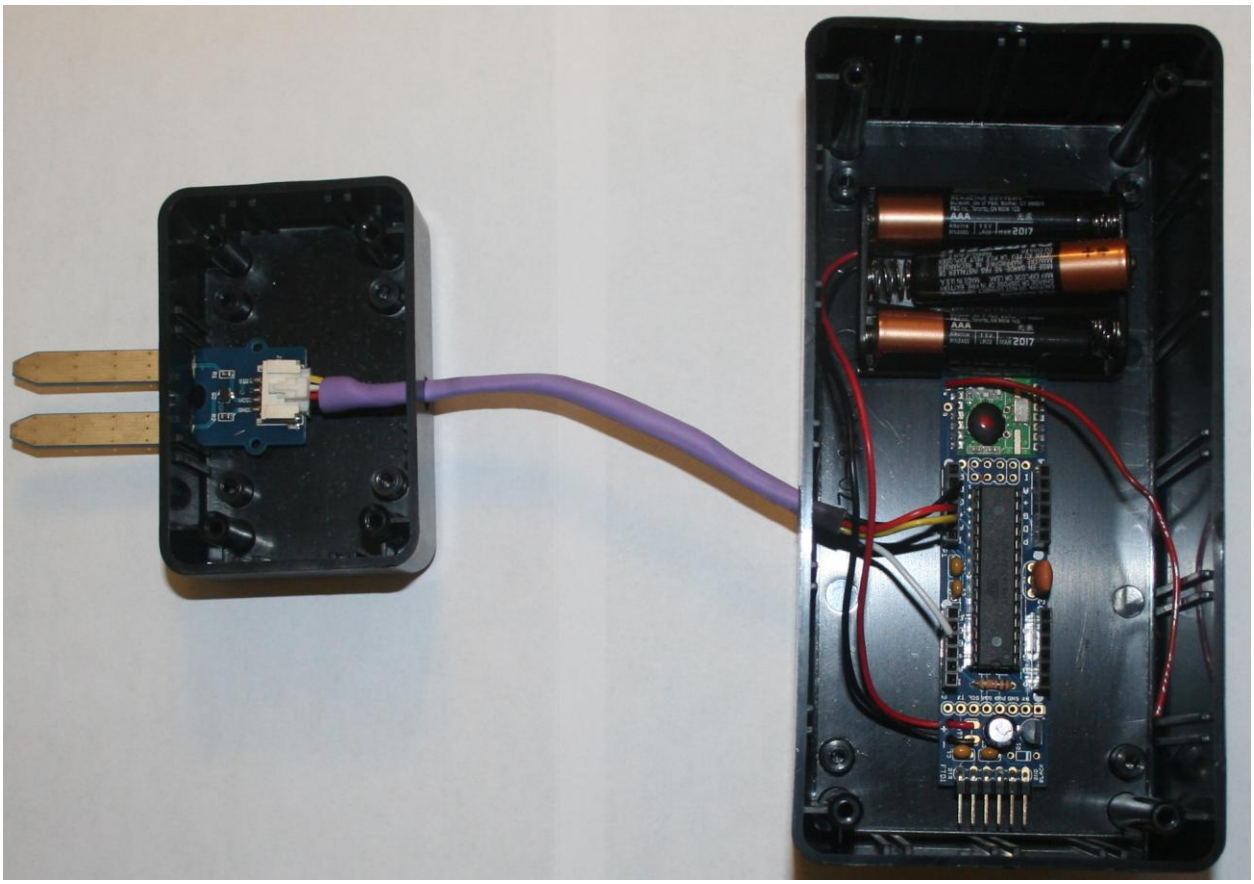

Figure 2: Assembled Sensor Node

Software Design

In designing the code for the project, I knew each node would need to report its sensor's value to the base station on a fixed schedule. This value would need to be read from the groove moisture sensor using one of the analog inputs of the ATMega exposed on JeeNode's headers. Additionally, I wanted to handle time synchronization so the node's reports would be consistent and would not drift too much. Finally, I wanted a node that was far from the house to be able to relay its findings through a node that was in proximity of the base station.

To accomplish this, I wrote two sets of code. The first runs on the JeeLink base station ATMega and does two things: first, it broadcasts a message once per second, which contains a variable that counts down. When the variable reaches 0, all nodes are expected to report their values. Second, it runs in a loop listening for incoming messages. For all messages sent and received, the JeeLink writes a line out to its serial interface, which is connected to the PC via USB.

The second program runs on both JeeNodes. When the node is powered on, it goes into a listening loop waiting for a message that contains a timestamp. It will spend at most 2 seconds waiting for a message from the base station or from another node that contains a timestamp from the base station. In testing, it was found that the message one-way trip time was consistently 6ms, so instead of using send/reply time synchronization like NTP, 6ms is subtracted from the supplied timestamp if it was sent by the base station, or 12ms if it was forwarded on from an intermediate node.

Once the node has gotten the timestamp or times out, the node will broadcast its current readings. If it had also received a timestamp from the base station, it includes this in the message, and sets a flag in the message that this time came from the base station.

After sending the readings, it will listen for another 2 seconds for messages from any other nodes. The node does know how many other nodes are expected in the system through a compile-time constant, so it can exit the loop before the full 2 seconds if it hears from all other nodes. If the node has gotten any reports from other nodes since it reported in, it combines all of them and broadcasts it as one message. This allows messages from nodes that are remote to the base station to be forwarded on. This design only allows for 2-hops (node -> node -> base station), but this is sufficient given the range of the radios and the size of my property.

Below is the layout of the message packets. It does not include the built in header provided by the JeeNode RF12 library, which handles adding packet size, node ID, and some flags not needed for this project.

Type	Name	Purpose
Int array (size is # of nodes in system)	Value	The sensor value for each sensor node. Unknown values from other nodes for this round are set to -1.
Unsigned long	Ticks	The count-down timer tick count
Boolean	syncBase	Set to true if this node has gotten a timestamp this round

Table 1: Packet Format

Once any final messages have been sent, the node calculates how long it has been awake by comparing its current tick count to the tick count when it started/ last woke. It then will go into a low power sleep mode until the next base station reporting time. The amount of time to sleep is calculated as the reported timestamp milliseconds minus the time it spent awake before beginning again. Ten seconds was chosen for the reporting interval, as it makes testing the node easy, however a much longer value could be used in production to minimize battery drain since moisture measurements do not change significantly over short periods.

Below is a Visio diagram of two nodes communicating with the base station when one is too far to report directly with the base station.

Figure 3: Data Transfer Sequence

Source Code

Software for JeeLink base station:

```
#include <JeeLib.h>
#include <SimpleTimer.h>

#define LED_PIN     9 //Used to link LED on TX/RX

//Number of sensor nodes in use, do not include base station
#define NUMBER_OF_NODES  2
#define BASE_ID  26
#define DELAY  10000
#define TICK  1000

struct {
  int value[NUMBER_OF_NODES];
  unsigned long ticks;
  boolean syncBase;
} payload, tickPayload;
```

```
SimpleTimer timer;

int countdown;

static void serialFlush ()
{
    #if ARDUINO >= 100
        Serial.flush();
    #endif
}

void timerWorker() {
  digitalWrite(LED_PIN, 0);
  countdown--;
  Serial.print(millis(), DEC);
  Serial.print(" Timer Worker:  ");
  Serial.println(countdown ,DEC);
  serialFlush();

  tickPayload.ticks = countdown*1000;
  tickPayload.syncBase = true;
  while (!rf12_canSend())
    getPkt();
  rf12_sendStart(0, &tickPayload, sizeof tickPayload);
  if(countdown==0)
    countdown = DELAY/1000;
  digitalWrite(LED_PIN, 1);
}

void setup ()
{
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, 1);
  Serial.begin(57600);
  Serial.println("\n[Link]");
  countdown = DELAY/1000;
  serialFlush();
  rf12_initialize(BASE_ID, RF12_915MHZ, 4);
  digitalWrite(LED_PIN, 0);
  timer.setInterval(TICK, timerWorker);
}

void loop ()
{
  timer.run();
  getPkt();
  digitalWrite(LED_PIN, 1);
}

void getPkt()
{
  while(rf12_recvDone() && rf12_crc == 0)
  {
    digitalWrite(LED_PIN, 0);
    memcpy(&payload, (void*) rf12_data, sizeof payload);
    int ID = rf12_hdr & 0x1F;
    if(ID > 0 && ID <= NUMBER_OF_NODES)  // Sanity check
    {
      Serial.print(millis(), DEC);
      Serial.print(", ");
      Serial.print(ID, DEC);  //Remote Node ID
      for(int i=0; i< NUMBER_OF_NODES; i++)
      {
        Serial.print(", ");
```

```
        Serial.print(i, DEC);
        Serial.print(":");
        Serial.print(payload.value[i], DEC);      //Value
      }
      Serial.print(", T:");
      Serial.print(payload.ticks);
      Serial.print(", S:");
      Serial.println(payload.syncBase);
      serialFlush();
    }
  }
}
```

Software for JeeNode:

```
#include <JeeLib.h>
#include <avr/sleep.h>

//Node ID (1 based)
#define NODE_ID  1

//Number of sensor nodes in use, do not include base station
#define NUMBER_OF_NODES  2

//ID of base station
#define BASE_ID  26

//Reporting interval
#define DELAY  10000

//Time to wait for reports from other nodes
#define MAX_WAIT  2000

struct {
  int value[NUMBER_OF_NODES];
  unsigned long ticks;
  boolean syncBase;
} payload, readPayload;

ISR(WDT_vect) { Sleepy::watchdogEvent(); }

void setup ()
{
  for(int i=0; i< NUMBER_OF_NODES; i++)
  {
      payload.value[i] = -1;
  }
  payload.syncBase=false;
  rf12_initialize(NODE_ID, RF12_915MHZ, 4);
}

void loop ()
{
  unsigned long start = millis();
  int reports = 0;
  payload.value[NODE_ID-1] = analogRead(A0);  //Store the value in the array: -1 to convert from 1-based to
0-based

  rf12_sleep(RF12_WAKEUP);
  delay(3);           // Give the radio time to wake up, otherwise the message could be corrupt

  boolean sync = false;
  unsigned long listenStart = millis();
  unsigned long timeout = 0;
  unsigned long ticks = 0;
```

```
unsigned long TxDelay = 6;
while(!sync && timeout < MAX_WAIT)  //Try to get a packet that has a timestamp
{
  if (rf12_recvDone() && rf12_crc == 0)
  {
    int ID = rf12_hdr & 0x1F;
    memcpy(&readPayload, (void*) rf12_data, sizeof readPayload);
    if(ID==BASE_ID)
    {
      sync=true;
      payload.syncBase=true;
      ticks = readPayload.ticks;
      payload.ticks = ticks;
    }
    else  //Cache results from any other nodes
    {
      if(readPayload.syncBase && !sync)
      {
        sync=true;
        payload.syncBase=true;
        ticks = readPayload.ticks;
        payload.ticks = ticks;
        TxDelay = 12;
      }
      reports++;
      for(int i=0; i< NUMBER_OF_NODES; i++)
      {
        if(readPayload.value[i] >= 0 && i!=(NODE_ID-1))
          payload.value[i] = readPayload.value[i];
      }
    }
  }
  timeout = millis() - listenStart;
}

TrySend();

boolean gotNewData = false;
listenStart = millis();
while(timeout < MAX_WAIT && reports < NUMBER_OF_NODES) //Try to get any packets from other nodes
{
  if (rf12_recvDone() && rf12_crc == 0)
  {
    int ID = rf12_hdr & 0x1F;
    memcpy(&readPayload, (void*) rf12_data, sizeof readPayload);
    if(ID!=BASE_ID)  //Cache results from any other nodes
    {
      gotNewData = true;
      reports++;
      for(int i=0; i< NUMBER_OF_NODES; i++)
      {
        if(readPayload.value[i] >= 0 && i!=(NODE_ID-1))
          payload.value[i] = readPayload.value[i];
      }
    }
  }
  timeout = millis() - listenStart;
}

if(gotNewData)
  TrySend();

delay(3);
rf12_sleep(RF12_SLEEP);
payload.syncBase = false;
```

```
  for(int i=0; i< NUMBER_OF_NODES; i++)
  {
      payload.value[i] = -1;
  }

  unsigned long wait=0;
  if(ticks==0)
    wait = DELAY-(millis()-start);
  else
    wait = ticks-(millis()-start);

  wait = wait - TxDelay;

  delay(wait%16);  //Burn any ms not divisible by 16
  wait = (unsigned long)((unsigned long)(wait / 16)*16);  //Sleep only works for increments of 16
  Sleepy::loseSomeTime(wait);
}

void TrySend()
{
  //Time to report in
  while (!rf12_canSend())
  {
    rf12_recvDone();
  }
  rf12_sendStart(0, &payload, sizeof payload);  // Ints are 2 bytes (16 bit microprocessor)
  rf12_sendWait(1);
}
```

Testing

Test 1: Moisture Sensor Readings

To get an idea of the possible values, I measured the values for different conditions using both sensors. The sensors showed a slight variation in sensitivity.

Condition	Sensor 1	Sensor 2
Dry (sitting on desk)	0	0
Dry Soil	119	136
Wet soil	432	459
Submerged (in cup of water)	473	516

Figure 4: Moisture Sensor Readings

Test 2: Radio Range

Next, I performed a series of range tests. All tests performed inside my house showed the radios were able to reach from one end of the house to the base station on the other. I then tested the units outside with the base station (JeeLink) connected to my PC sitting in my office. The furthest I could go outside the house before experiencing occasional packet loss was 28 feet from the PC (through an interior plus an exterior wall). I then tested having one node outside at 25 feet, with the second using it as an intermediary. The secondary node was received reliably by node one when it 57 feet from node one (line of sight, no obstructions). This made node two 82 feet from the base, which is sufficient to cover my yard. I did not measure the maximum distance node two could be from node one because moving beyond my property would result in climbing through brush, standing in the street, crossing a creek, and/or trespassing.

Test 3: Battery Life & Power Draw

The JeeNode uses the ATMega328P instead of the ATMega168 that is common on many Arduinos. The 328P is the pico-power (ultra-low power draw) version of the microcontroller. To measure the potential battery life of the unit, I first measured the power draw of a node as it ran through the code sequence using a multi-meter:

Event	Approximate Current Draw
Sleeping	0.11 mA
Awake, running code	7.6 mA
Awake, sending/receiving packet	17.9 mA

Figure 5: Current Draw

Considering the unit will spend at most 4 seconds awake, of which only a fraction is sending/receiving, and the remainder of the reporting interval (currently 10 seconds) in the low power state, the unit does not use much power per cycle. At most, it might consume about 40mA per cycle, though more likely it would use approximately:

- 17.9 mA for 6ms per send/receive – with at most 5 sends/receives per interval = 17.9 mA for 30ms (.03 seconds)
- 7.6 mA for 2 seconds while running the code and waiting messages = 7.6 mA for 2 seconds
- 0.11 mA for 10 seconds - 2.03 seconds = 0.11 mA for 7.97 seconds

This equals:

mA	Sec	Draw (mA)	
17.9	* 0.03 =	0.537	
7.6	* 2 =	15.2	
0.011	* 7.97 =	0.08767	
	=	15.82467	mA/cycle
	=	**1.58**	**mA/second**

As an additional test, I left a node reporting for 3 days. The batteries started reading at 1.535v each (4.6v in series). After 3 days of updates every 10 seconds, they were at 1.3v (3.9v overall). According to detailed test results from JeeNode's creator[14], the nodes will continue to function down to 1.85v. For this setup, it works out to:
4.6v – 3.9v = .7v used in 3 days = .233v used per day
4.6v – 1.85v = 2.75v useful range.
2.75v / .233 v = 11.8 days of run time.

The datasheet[15] for the AAA batteries in use do not show the service hours for a 1-2 mA discharge rate, but at 10mA, the batteries have a 100-hour service life. Additionally, these numbers are not linear as a 250mW draw drops to 0.8v within 3 hours, but 100mW does not hit 0.8v for nearly 12 hours. Based on these charts, it is possible the nodes could run considerably longer than 12 days.

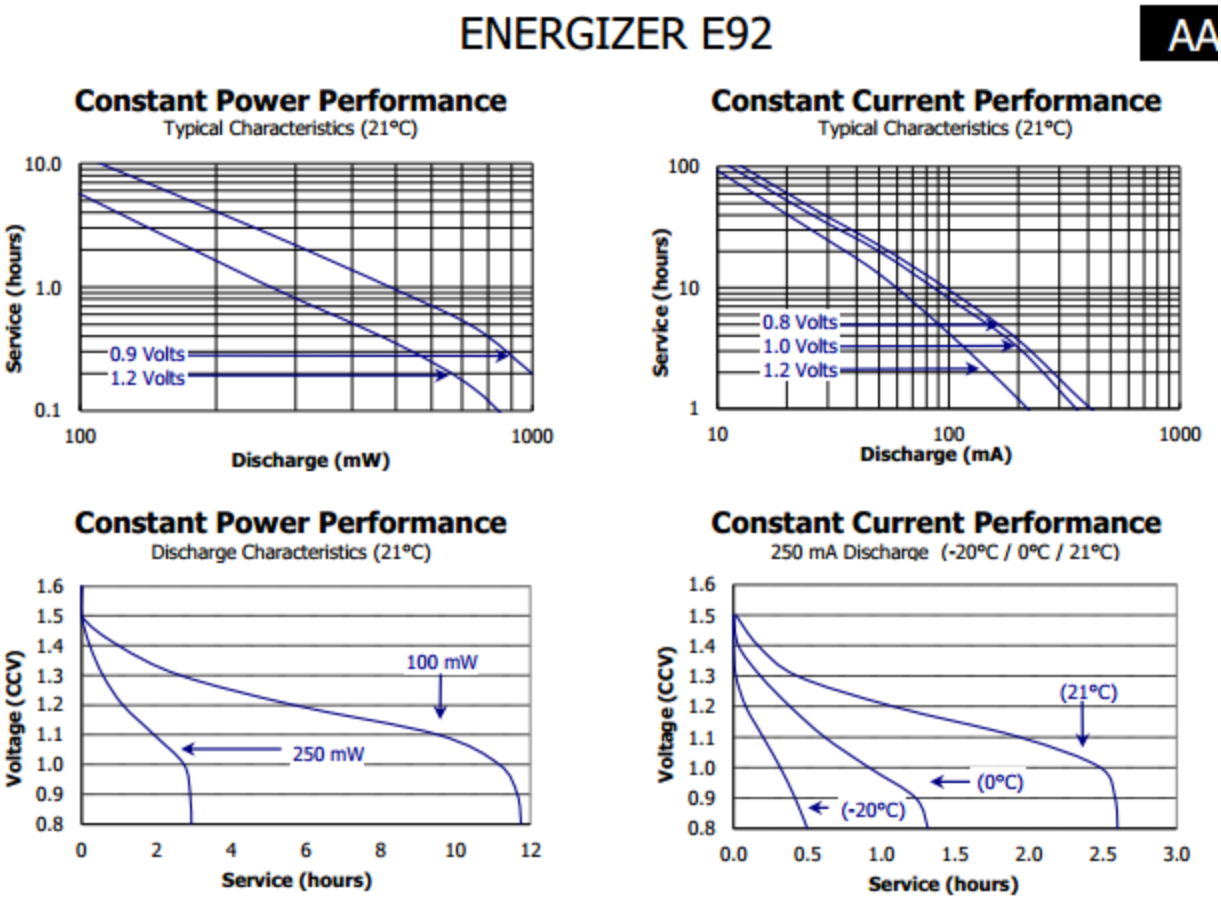

Figure 6: AAA Battery Manufacturer Performance Specifications

Test 4: Time Synchronization-Message Passing

Nodes consistently reported in 12ms-13ms after the base station sent its tick. Given that the message transmission delay is 6ms each way, this is an expected delay. A full sequence of both nodes getting a time sync and replying, then getting each other's replies and replying again appears in the console as:

```
40001 Timer Worker:  0
40012, 1, 0:0, 1:-1, T:0, S:1
41001 Timer Worker:  9
41013, 2, 0:-1, 1:0, T:9000, S:1
42001 Timer Worker:  8
42012, 1, 0:0, 1:0, T:0, S:1
43001 Timer Worker:  7
43013, 2, 0:0, 1:0, T:9000, S:1
```

(Note that both nodes were not in water at the time so the correct sensor values are 0.) The overall base station tick count is first, followed by either the timer count down until the next report interval, or the message from a node in the format (node ID), (value at node 1), (value at node 2), (timestamp), (if the timestamp was sync'd). In this sequence, it can be seen that node 1 sends its value after getting timestamp 0 from the base station (S is 1, indicating it received and 12ms is the approximate round trip time for the packets.) Node 2 had to delay sending as the communication channel was in use by node 1, but got the next timestamp from the base and set its timestamp to 9 seconds and sent its reply. Node 1 then saw node 2's reply and rebroadcast it a second later (both values now 0, neither are -1). Finally, node 2 node 1's broadcast and rebroadcast it.

Here is a sequence when node 1 was outside of the range of the base station, but node 2 picked up its broadcast and rebroadcast it:

```
1250001 Timer Worker:  0
1250013, 2, 0:-1, 1:0, T:0, S:1
1251001 Timer Worker:  9
1252001 Timer Worker:  8
1252013, 2, 0:68, 1:0, T:0, S:1
```

This shows node 2 successfully reading node 1's sensor as 68 (note the labels are 0-based so 0:68 is from node 1).

Potential Improvements

There are a number of improvements that could be made to increase the functionality, reliability, and usability of the system. I'll briefly discuss a few here.

A number of additions could be added to the nodes to make the system more functional. Light[16], wind[17], temperature[18], barometric pressure[19], and (air) humidity[20] sensors could easily be added to the nodes for additional cost. These would provide a more complete picture of each plant's environment. The JeeNodes each have four analog inputs and four digital IO pins, so more sensors could be accommodated (or an additional 16 using an expansion board[21]). The only change in the software needed would be to read from the additional inputs and to expand the message struct to include additional value arrays. Further, if the range of the base station plus one intermediary node and a remote node is not sufficient, the message algorithms chosen could be modified to support more than one intermediary hop to the base station. Alternatively, a different radio with more range could be used.

To add to the reliability of the system, a few other components could be added. JeeLabs makes a power supply board[22], which would increase usable life of the AAA batteries used. Alternately, the batteries could be replaced with rechargeable batteries and a solar panel could be added with a charging circuit to minimize battery waste in the system. Finally, if the preciseness of the current time synchronization method was unacceptable to some users, either a more precise Real Time Clock board[23] could be added, or a GPS module could be added and provide time keeping.

Finally, to improve the usability of the system, a few changes could be made. First, instead of hard coding the number of nodes and the node ID in software, two potentiometers could be added to each node, and the node could read these values upon waking so the node network could be changed on the fly in the field. Second, a longer cable between the sensor and the microcontroller would be helpful. Third, a longer sleep period could be used to increase battery life of the units. Last, a GUI app on the PC to read from the console output and show the values in a friendly way (perhaps on a map) would be a good improvement.

Conclusion

Overall, I am pleased with the results of this project. The nodes reliably report their readings to the base station and it is easy to see if a plant is in need of water. In addition, it was interesting to troubleshoot wireless communication issues and to consider the best way to handle exchanging information when power consumption was a key concern. As a first attempt at using wireless communication with microcontrollers, it was exciting to be able to get information from an Arduino that was not tied to a PC.

---

[1] http://arduino.cc/en/Main/ArduinoBoardNano/
[2] http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-series1-module#overview
[3] http://www.robotshop.com/arduino-nano-xbee-board-2.html
[4] http://jeelabs.org/
[5] http://shop.moderndevice.com/products/jeenode-kit
[6] http://www.hoperf.com/rf_fsk/fsk/20.htm
[7] http://shop.moderndevice.com/products/jeelink-module-fully-assembled
[8] http://www.seeedstudio.com/depot/grove-moisture-sensor-p-955.html

[9] http://www.radioshack.com/product/index.jsp?productId=2102735
[10] http://www.seeedstudio.com/depot/grove-universal-4-pin-20cm-cable-5-pcs-pack-p-749.html?cPath=178_179
[11] http://www.radioshack.com/product/index.jsp?productId=2062283
[12] http://www.radioshack.com/product/index.jsp?productId=2062279
[13] https://www.sparkfun.com/products/9717
[14] http://jeelabs.org/tag/lowpower/
[15] http://data.energizer.com/PDFs/E92.pdf
[16] http://shop.moderndevice.com/products/ambi-light-sensor
[17] http://shop.moderndevice.com/products/wind-sensor
[18] http://shop.moderndevice.com/products/tmp421-temperature-sensor
[19] http://jeelabs.com/products/pressure-plug
[20] http://shop.moderndevice.com/products/humidity-and-temperature-sensor
[21] http://jeelabs.com/products/input-plug
[22] http://jeelabs.com/products/aa-power-board
[23] http://jeelabs.com/products/rtc-plug